POLITECNICO
MILANO 1863

Computer Science and Engineering

PowerEnjoy Service - Design Document

December 5, 2016

Prof. Luca Mottola

Authors:

- ZHOU YINAN(Mat. 872686)
- ZHAO KAIXIN(Mat. 875464)
- ZHAN YUAN(Mat. 806508)

Contents

1 INTRODUCTION

1.1 Purpose

The Design Document serves to describe the structure of the PowerEnJoy service. It provides all the detailed information for building the system. More precisely, it provides the information on the chosen architecture style and design pattern. It explains how the components are implemented and interacted in the system. The document also provides the verification of fulfilling the requirements listed in the RASD documents. In general, the document is served as a guideline for program developers.

1.2 Scope

The Design Document shows how the system is built and explains how the functional requirements in the RASD file are realized. The document covers high level architecture design, interacting components, algorithms and user interface design.

1.3 Definitions, Acronyms, Abbreviations

- RASD : Requirement Analysis and Specification Document
- DD : Design Document
- MVC : Model View Controller
- REST : Representational state transfer (REST) or RESTful web services are one way of providing interoperability between computer systems on the Internet.
- JSON : JSON (JavaScript Object Notation) is a minimal, readable format for structuring data.

1.4 Reference Documents

- Specification Document Assignments AA 2016-2017
- RASD

1.5 Document Structure

The Design Document is divided into 7 parts :

- **Introduction** : This section introduces the structure of Design Document and some basic background knowledge to understand the document.

- **Architecture Design**
  1. High level components and their interactions : This section gives a general description of how the components are defined and how they communicate with each other
  2. Component view : This section gives detailed information of components defined in the system
  3. Deployment view : This section describes how the components are deployed in order to act correctly
  4. Runtime view : This section gives the sequential diagrams of how the users accomplish their requests
  5. Component interfaces : The interfaces of components are described in this section
  6. Selected architectural styles and patterns : This section explains the reason of choosing certain architecture and the benefits accompanied
  7. Other design decisions

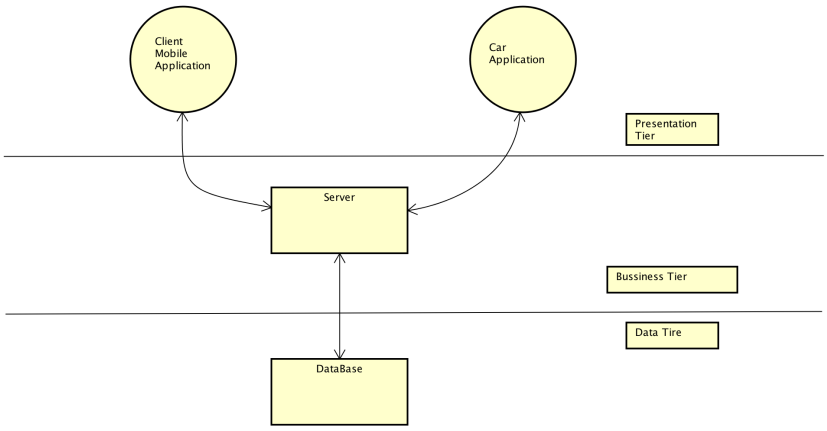
- **Algorithm Design** : In the section, the necessary code samples are given in order to clearly demonstrate the component interaction

- **User Interface Design** : this section presents mockups and user experience explained via UX and BCE diagrams

- **Requirement Traceability** : This section shows how the requirements in the RASD are accomplished in the design

2 ARCHITECTURAL DESIGN

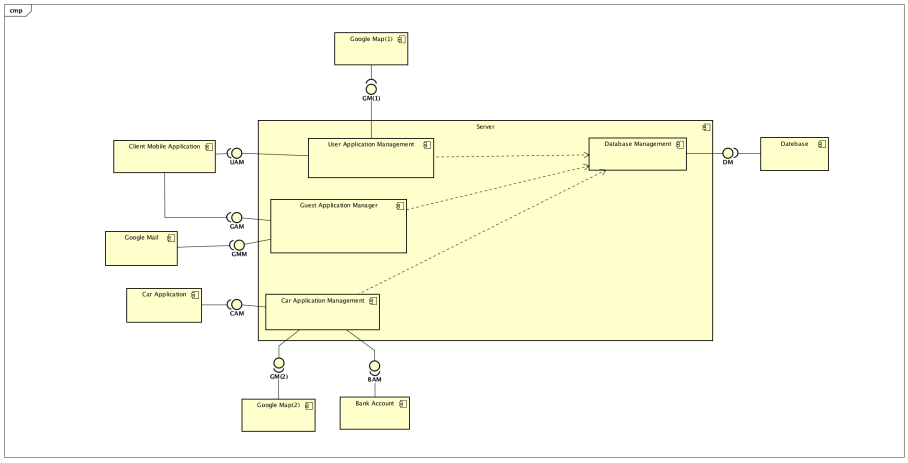
2.1 High level components and their interaction



Here we choose to use the 3 tier architecture: The Presentation tier includes the Client Mobile Application, Client Web Browser and the Car Application; The Business tier includes the Server; The Data tier includes the database.

Here a MVC pattern is also used. The presentation tier is responsible for all the GUI part, thus it is the view part. The server maintains all the business logic and the database is responsible for data manipulation. Thus the model and controller are in the server and database side.

2.2 Component view



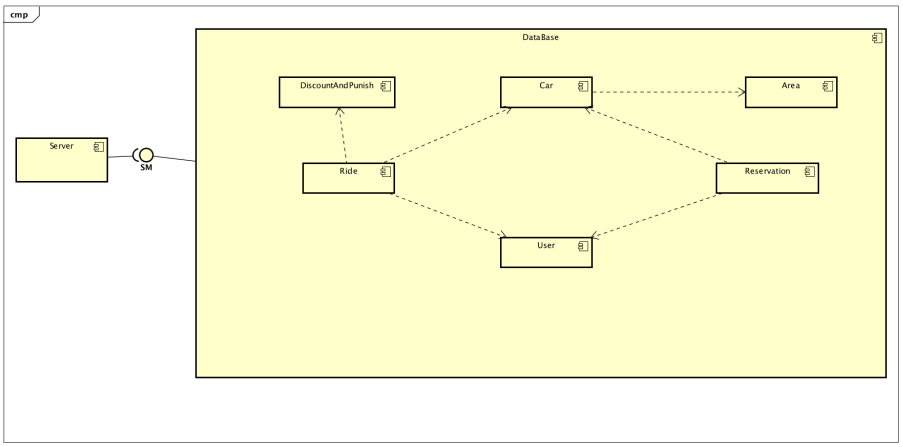
For the Server, there should be several interfaces, which are in charge of communicating with other components:

- **User Application Manager** : Component that provides the interface for the users of the system and the Google Map. User Application Manager should provides the options of checking the available car around the specified position, reserving car and canceling the reservation. With the help of the Google Map, the system can know the location of the user.

- **Guest Application Manager** : Component that provides the interfaces for the unregistered guest of the system and the Google Mail. It provides the option to create an account. Finishing the register, the system would send an e-mail to the guest through the Google Mail.

- **Car Application Manager** : Component that provides the interface for the car Application, Google Map and Bank Account. The Car Application Manager can know the location of the car, charge the specified bank account and send the instruction to the car.

- **Database Manager** : Component that provides the interface for the Database for collecting the information which are collected by other components.



For the Database, there are several component for storing the data of the system. The Database should provides the interface of the Server for information exchange. Through the interface, the Server can transmit and store the information in the Database. The Server can also retrieve the information from the Database according to the corresponding instruction.

- **Car** : Component that stores the information of the cars. The Car should stores the position, plate, battery, capacity, state and the currentPosition of each car.
- **User** : Component that stores the information of the users. The User should stores the credential and the paymentInfo of each user.
- **Ride** : Component that stores the information of the riding. The Ride should stores the riding information, including the renting car, corresponding user and so on.
- **Reservation** : Component that stores the information of the reservation. The Reservation should stores the user and the corresponding car of each reservation.
- **DiscountAndPunish** : Component that stores the information of the discount and punish of each ride. For each ride, when finishing the riding, the DiscountAndPunish component should provide the information about the discount and the punish, for calculating the total charging.
- **Area** : Component that stores the area of each position.

2.3 Deployment view

2.4 Runtime view

You can use sequence diagrams to describe the way components interact to accomplish specific tasks typically related to your use cases.

2.5 Component interfaces

2.5.1 Client app

register(name, surname, email,drivingLisence,paymentInfo)

The register function on the Client app side receives from input all the necessary information for registering. This information is then sent to the server in the format of JSON for former processing.

logIn(email, password)

The logIn function receives from the input the user's email and corresponding password. This information is then sent to the server in the format of JSON for former processing.

car[] getAvailableCars(location)

By providing a location, users can get all the available cars nearby. The location information is sent to the server in the format of JSON then the server sends back all the cars that are available near that location.

void reserve(user, car)

The reserve function allows a user to reserve a car. The reserve request is then sent to the reservation Manager in the server.

void openTheDoor(car,location)

When the user gets to the reserved car, he can inform the server that he is nearby by sending his current location to the server.

2.5.2 Car app

void startRide(user, car)

Car app notifies the server that a ride starts.

void endRide(user, car, state)

Car app notifies the server that a ride is finished.

String getCurrentPrice(user,car)

The working car can get the current price information from the server.

2.5.3 Guset Manager

Boolean register(credentials)

The register function receives from clients the credentials. The Guest Manager in the server checks whether the received credentials already exist in the database or not. In the case of new coming credentials, the Guest Manager uses Gmail API to send an email to the guest with a newly generated password. Then Guest Manager creates a new tuple using credentials and password and inserts it into the database. In the case of a successful register, a true value will be sent to the cilent app. In the case of non valid credentials, the server sends back a false notation to the client app.

String signIn(email, password)

The signIn function checks the email and password provided by the user in the server side. If they are valid i.e. there exists a corresponding email and the password matches the email, then the information of the user will be encoded into a JSON format and be sent back to the client app.

2.5.4 User Manager

String reserver(user, car)

User makes request to the server for a car reservation. User Manager processes the request and makes reservation to the corresponding car.

The reservation information is then stored into database and the corresponding car information will be sent back to client app in the format of JSON.

Boolean reservationTimeOut(user, car)

When the reserved car is timing out, User Manager notifies the corresponding user and cancels the reservation.

2.5.5 Car Manager

Ride startRide(user,car)

Car Manager on the server side receives from car app the user and car information. It checks from the database whether the ride request is valid or not. In the case of a valid request, server sends back to the client app a successful notation. Also Car Manager stores the data into the database. In the case of a non-valid request, the server sends back to the car app an error notation.

String endRide(user,car,state)

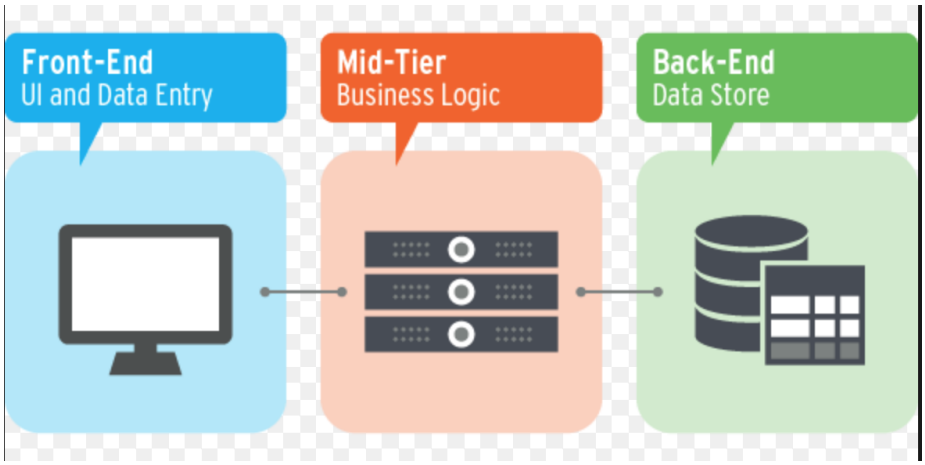
Car Manager processes the request of ending a ride from the car app. It gets the ride information from the database and calculates the final price of the ride. Then Car Manager sends back to car app the price needs to pay. Car Manager redirects payment request to the bank service.

2.6 Selected architectural styles and patterns

2.6.1 Architecture

We choose the three-tier architecture for our system. The whole system is divided into three parts : presentation tier, business tier and data tier.

- presentation tier : It contains Client app and Car app. It is the layer that directly interacts with the actors.
- business tier : Business tier contains all the logic. It manages request and response. Also it manages the data interaction with the database. By using the business tier, clients do not directly interact with the database.
- data tier : We use MySQL database to store all the useful data.



The three tier architecture is used because an effective distributed client-server design is needed. It guarantees increased performance, flexibility, maintainability, reusability, and scalability, while hiding the complexity of distributed processing from the user. In our system, no direct database access from the client app is allowed. All the requests are passing through business tier first.

We use a typical client-server architecture. In the client-server architecture, data is manipulated in the server side and can only be accessible to designated users. Moreover, services can be easily maintained or updated in the server side without touching Client app.

The Client app is composed of a mobile application and a web browser. Users can choose to access the service by either a mobile web browser or an application. Our system uses request-response methodology. All the services are requested from the user side. The client asks for information or requests for certain services using REST and server responses in JSON format. When the server receives a certain request, it dispatches the request to corresponding responsible manager.

We use RESTFUL API for for our service to be processed. Each service is defined by a unique URI. By using RESTFUL API, we keep our client-server communication clear and simple. Moreover, REST style client-server architecture allows stateless session in the server. All the information transfered between server and client side is in JSON format which allows the information to be processed by different end-systems.

2.6.2 Pattern

MVC Model-View-Controller is used in our system. To combine it with our three-tier architecture, the view part lies on the client app side which deals with interaction with users. Model and Controller are in the server side, which maintains business logic and manipulation of data.

Client-Server In our system, there are 2 clients: user application and car application. The user application can be a pure iOS or Android app or it can be a mobile web browser. In both cases, we use RESTFUL API to communicate with the server. The information sent and received is in JSON format.

Request-response Our system is based on service which uses request-response methodology. All the requests are generated from the user side which can be either a user application or a cat application. Server is responsible for processing the request and sending back the response.

2.7 Other design decisions

3 ALGORITHM DESIGN

Focus on the definition of the most relevant algorithmic part

4 USER INTERFACE DESIGN

This section will describe different interfaces for the components. Component interfaces are divided into three different aspects : Guest Manager, User Manager and Car Manager.

5 REQUIREMENT TRACEABILITY

The design of architecture need to fulfill all the goals and requirements requested in the RASD file. Here readers can find how these specifications are completed using our component interfaces.

6 EFFORT SPENT

7 REFERENCES